An efficient algorithm to find a set of nearest elements in a mesh

Gleb Novitchkov

email: gleb.novitchkov@gmail.com

October 4, 2011

Contents

1 Introduction

Here we present an algorithm that find a list of elements neighboring some given element in a linear time. More precisely, if there are N_{elem} elements in the mesh, the runtime of the algorithm is $\mathcal{O}(N_{\text{elem}})$.

2 Definition

Definition 2.1. By *element* we mean a 3-simplex Δ^3 imbedded in $\mathbb{R}^3$.

Essentially an element is a tetrahedron in $\mathbb{R}^3$.

Goal: given an element of the mesh, we want to find a set of elements that are near this element.

Definition 2.2. Given an element E, we call an element F a **vertex-near** element of E if E and F share common vertex; we call an element F a **edge-near** element of E if E and F share common edge; we call an element F a **face-near** element of E if E and F share common face. An element F is **near** element of E if F is either face-near, edge-near, of vertex-near element of E.

2.1 Mesh and the representation of the elements

A mesh is specified by the cloud of n points, or *nodes*, $\{p_0, p_1, \ldots, p_{n-1}\}$, where $p_i = (x_i, y_i, z_i)$. An element E_i is specified by the four nodes, $E_i = \{p_{i_0}, p_{i_1}, p_{i_2}, p_{i_3}\}$.

3 The algorithm

The idea of the algorithm is to do the histogramming of the nodes.

Step 1. (initialization) For N_{node} nodes allocate array L of lists, the length of the array is N_{node}.

Step 2. (histogramming) For each element i_{elem}, $0 \leq i_{\text{elem}} \leq N_{\text{elem}} - 1$, do:

Step 2.1 For each node p_{j_i}, $0 \leq i \leq 3$ of element i_{elem} add index i_{elem} to the lists $L[j_0]$, $L[j_1]$, $L[j_2]$, $L[j_3]$. In C++ notation, it should be: `L[`j_0`].push_back(`i_{elem}`)`, `L[`j_1`].push_back(`i_{elem}`)`, `L[`j_2`].push_back(`i_{elem}`)`, `L[`j_3`].push_back(`i_{elem}`)`.

Step 3. (Finding neighboring elements) Given element $E_i = (j_0, j_1, j_2, j_3)$, list of elements neighboring E_i is given by the union of the lists $L[j_0]$, $L[j_1]$, $L[j_2]$, and $L[j_3]$.

Step 4. Deallocate array L of lists.

4 Parallelization

Algorithm admits easy parallelization by OpenMP, one only has to take care to use critical section for Step 2.1: adding element index to the lists should be done inside the critical section.

5 Conclusion

Missing details will be provided later.

An efficient algorithm to find a set of nearest elements in a mesh

Gleb Novitchkov

email: gleb.novitchkov@gmail.com

October 4, 2011

Contents

1 Introduction

Here we present an algorithm that find a list of elements neighboring some given element in a linear time. More precisely, if there are N_{elem} elements in the mesh, the runtime of the algorithm is $\mathcal{O}(N_{\text{elem}})$.

2 Definition

Definition 2.1. By *element* we mean a 3-simplex Δ^3 imbedded in $\mathbb{R}^3$.

Essentially an element is a tetrahedron in $\mathbb{R}^3$.

Goal: given an element of the mesh, we want to find a set of elements that are near this element.

Definition 2.2. Given an element E, we call an element F a **vertex-near** element of E if E and F share common vertex; we call an element F a **edge-near** element of E if E and F share common edge; we call an element F a **face-near** element of E if E and F share common face. An element F is **near** element of E if F is either face-near, edge-near, of vertex-near element of E.

2.1 Mesh and the representation of the elements

A mesh is specified by the cloud of n points, or *nodes*, $\{p_0, p_1, \ldots, p_{n-1}\}$, where $p_i = (x_i, y_i, z_i)$. An element E_i is specified by the four nodes, $E_i = \{p_{i_0}, p_{i_1}, p_{i_2}, p_{i_3}\}$.

3 The algorithm

The idea of the algorithm is to do the histogramming of the nodes.

Step 1. (initialization) For N_{node} nodes allocate array L of lists, the length of the array is N_{node}.

Step 2. (histogramming) For each element i_{elem}, $0 \leq i_{\text{elem}} \leq N_{\text{elem}} - 1$, do:

Step 2.1 For each node p_{j_i}, $0 \leq i \leq 3$ of element i_{elem} add index i_{elem} to the lists $L[j_0]$, $L[j_1]$, $L[j_2]$, $L[j_3]$. In C++ notation, it should be: `L[`j_0`].push_back(`i_{elem}`)`, `L[`j_1`].push_back(`i_{elem}`)`, `L[`j_2`].push_back(`i_{elem}`)`, `L[`j_3`].push_back(`i_{elem}`)`.

Step 3. (Finding neighboring elements) Given element $E_i = (j_0, j_1, j_2, j_3)$, list of elements neighboring E_i is given by the union of the lists $L[j_0]$, $L[j_1]$, $L[j_2]$, and $L[j_3]$.

Step 4. Deallocate array L of lists.

4 Parallelization

Algorithm admits easy parallelization by OpenMP, one only has to take care to use critical section for Step 2.1: adding element index to the lists should be done inside the critical section.

5 Conclusion

Missing details will be provided later.